Kaczur Sándor*[870]*:* *Lekérdezések és adatfeldolgozási algoritmusok optimalizálási lehetőségei Java SE és Oracle HR környezetben

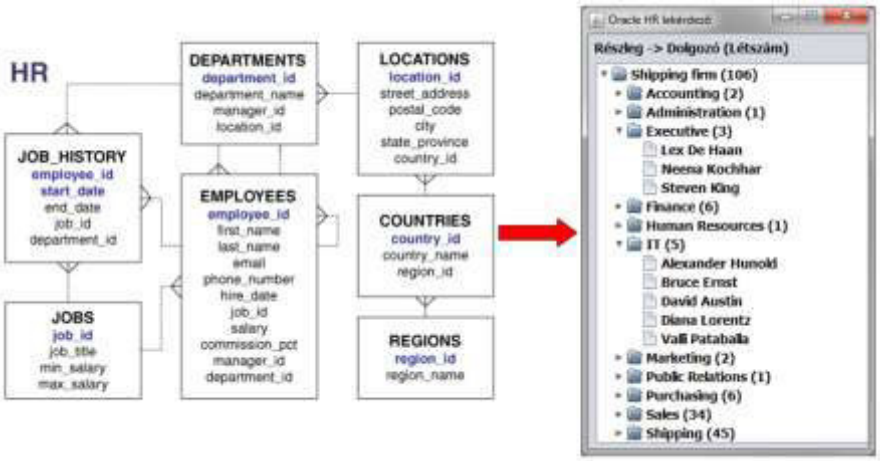
Absztrakt: Az Oracle HR séma minta-adatbázisa kiváló környezetet biztosít az adatbázisbeli lekérdezések és adatfeldolgozási algoritmusok működésének tesztelésére. A probléma leképezésének szintjétől és az elosztott alkalmazás-logika egyensúlyától függően más-más a terhelés célpontja, eltérő az adatbázis-szerver és a kliensprogram közötti kommunikációs adatforgalom mennyisége, eltérő lehetőségek adódnak az adatfeldolgozásra. Kiindulva egy okos lekérdező parancsból, a keletkező eredménytábla adatain és metaadatain végzett értelmezésből, illetve az algoritmikus utófeldolgozásból, hasznos tapasztalatokra tehetünk szert. A cikk összefoglalja a technológiai lehetőségeket az Oracle HR séma minta-adatbázisán alapuló Java SE Swing grafikus felhasználó felülettel rendelkező kliensprogramhoz kapcsolódóan. MVC-ben gondolkodva eltérő modellbeli adatszerkezetek alkalmazásával szintén fejleszthetünk hatékonyabbnak tekinthető alkalmazást. Az OJDBC driver képességeit, a java.sql csomag szolgáltatásait felhasználva és különböző vizuális komponensek mögötti adatmodellek közötti kapcsolatra építve a lekérdezések és adatfeldolgozási algoritmusok eredményei a felhasználó számára egyszerűen használható GUI-n érhetők el.

Specifikáció

Ha szoftverfejlesztőként egy összeszokott csapatban dolgoznak, akkor az informatikusok félszavakból is értik egymást. Ha egy projektmunkára keresünk munkaerőt, akkor a tipikus kulcsszavak alapján egy jó vezető azonnal megtalálja az alkalmas kollégát a feladat megoldására (természetesen a szaktudáson kívüli minden körülményt figyelembe véve, pl.: kapacitás).

Elhangzó hívószavak, kulcsszavak lehetnek például: Java alapú fejlesztés, Oracle adatbázis-kezelő, Oracle HE séma minta-adatbázis [1], MVC tervezési minta [2], Java SE Swing GUI, OJDBC driver, java.sql csomag, hatékonyság.

Ha rendelkezésünkre áll a megrendelővel előzetesen egyeztetett GUI terv (mert például gyorsan elkészült egy MS Visio-val, vagy a NetBeans JFrame Form varázsló eszközével, akkor az informatikusok egymás között úgy is specifikálhatnák a feladatot, hogy „oldd meg az 1. ábrán látható feladatot”!



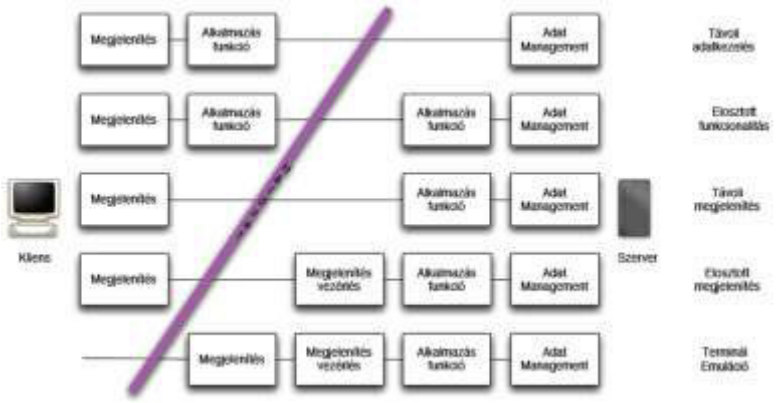
1. ábra: Specifikáció – informatikusok egymás közötti használatára

[870] Gábor Dénes Főiskola, kaczur@gdf.hu

Értelmezzük az 1. ábrát:

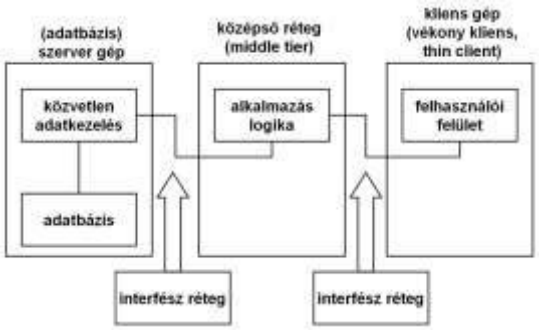
- Rendelkezésre áll az adatbázis EK diagramja.
- A GUI-t látva kiderül, hogy az adatbázisból a DEPARTMENTS és az EMPLOYEES táblák rekordjaira lesz szükségünk.
- A fastruktúrából (is) látszik, hogy a részlegek és az alkalmazottak között a kapcsolat foka 1:N. A két táblát a DEPARTMENT_ID attribútum köti össze.
- Közvetlenül kell a DEPARTMENT_NAME mező, konkatenálva a FIRST_NAME + " " + LAST_NAME mezőkkel.
- Megszámolni is kell: globális létszámösszesítés: Shipping firm (106), valamint lokális (részlegenként): például IT (5).
- Vegyük észre, hogy a fa gyökere fiktív, nem található meg az adatbázisban.
- A részlegek nevei ábécé sorrendben épülnek be a fába, közvetlenül a gyökérelem alá, és a részlegenkénti alkalmazottak nevei és ábécé sorrendben kerülnek levélelemként a megfelelő részleget alkotó csomópont alá.
- Ehhez a feladathoz elegendő az adatbázisból olvasni, nincsenek karbantartó műveletek.

Oldjuk meg a feladatot úgy, hogy a JDK 1.8 +NetBeans 8 fejlesztőeszközzel, fejlesztőkörnyezettel egyszerű Java Class Library típusú projektet hozunk létre. A projekthez adjuk hozzá az OJDBC drivert. A felhasználói felület nem kihívás, varázsoljuk össze a JFrame Form eszköz palettájáról. Most nem ez a hangsúlyos.



2. ábra: Elosztott alkalmazás architektúra

Ez egy funkcionálisan elosztott alkalmazás lesz, így világosan látni kell, hogy a fentiek közül (2. ábra, [3]) pontosan melyik szemléletben kell gondolkodnunk. A megjelenítés a kliens oldal feladata (javax.swing csomag JFrame, JPanel, JLabel, JTree vizuális komponensei), az adat menedzsment réteget az Oracle adatbázisszerver alkotja. Az alkalmazás funkciót megvalósító réteg elosztott, hiszen a kliens oldalon ez metódusokba szervezett alkalmazáslogikát jelent (java.sql csomag DriverManager, Connection, Statement, PreparedStatement, ResultSet és SQLException osztályai/interfészei), ennek szerver oldali részét pedig az OJDBC driver csomag szolgáltatási alkotják. Az elosztott alkalmazásfunkció közötti kapcsolattartást a hálózati réteg biztosítja.



3. ábra: Többrétegű alkalmazás felépítése

Más szemléletben ugyanez a 3. ábra [3]. Itt a középen található alkalmazáslogika két interfészen keresztül kapcsolódik a szerverhez (OJDBC driver és java.sql csomag együtt) és a klienshez (ez a felkínált 5 metódus egyike). Minden felhasznált szoftver ingyenes.

Az 5 különböző megvalósítás

Mind az 5 módszer implementációja más-más koncepció alapján valósul meg, végül ezen módszerek összehasonlításra kerülnek. A konstuktor hívja meg az 5 közül mindig azt a metódust, amelynek hatékonyságát tesztelni szeretnénk. Nem kezelünk helyben – a metódusok belsejében, ahol keletkezhet – kivételeket, hanem mindent továbbdobunk.

Az MVC (Model-View-Controller) kontra hatékonyság közül a hatékonyság vizsgálatára helyezzük a hangsúlyt, így az 5 metódus esetén kerevedni fog a modellhez, a nézethez és a vezérlőhöz tartozó forráskód, de ennek elvi hátránya a hatékonyság szempontjából gyakorlati előnyt jelent. Átmeneti adattárolásra – ha szükséges – POJO-kat használunk, amelyeket generikus java.util.ArrayList adatszerkezetekbe gyűjtünk össze. A JTree vizuális komponens mögötti javax.swing.tree. DefaultTreeModel adatmodellben javax.swing.tree.DefaultMutableTreeNode osztályú objektumok alkotják a fa gyökerét, csomópontjait és leveleit.

Az alkalmazáslogikát alkotó 5 metódus neve rendre method1(), …, method5(), minden kivételt továbbdobnak (throws SQLException, ClassNotFoundException). Mindegyik 3 deklarációjával kezdődik, url, user, password, amelyek az adatbáziskapcsolat felépítéséhez szükségesek. Mindegyik meghívja a driver osztálybetöltőjét is (4. ábra: 191-194. sor).

Több koncepció elvi alapját [4, 5] adja.

Az 1. módszer

A fa előkészítése a treeRoot gyökérelem deklarációjával kezdődik. Lényeges a Mutable tulajdonság, hogy utólag megváltoztatható legyen a treeRoot objektum userObject tuplajdonsága – hiszen még nem áll rendelkezésre a globális alkalmazotti létszám. A vizuális komponens mögötti adatmodellt a treeModell objektum alkotja, ebbe belekerül az előzőek szerint létrehozott gyökérelem (4. ábra: 196-197. sor).

Meg kell nyitni az adatbáziskapcsolatot és elő kell készülni egy későbbi SQL parancs lefuttatásához. Le kell kérdezni a részlegek nevét. A SQL1 szöveges változó tartalmazza az ehhez szükséges SELECT parancsot (4. ábra: 199-205. sor).

```
190  private void method1() throws SQLException, ClassNotFoundException {
191    String url="jdbc:oracle:thin:@localhost:1521:xe";
192    String user="HR";
193    String password="hr";
194    Class.forName("oracle.jdbc.driver.OracleDriver");
195    //fa előkészítése
196    DefaultMutableTreeNode treeRoot=new DefaultMutableTreeNode("Shipping firm");
197    DefaultTreeModel treeModel=new DefaultTreeModel(treeRoot);
198    //részlegek
       Connection connection=DriverManager.getConnection(url, user, password);
200    Statement statement1=connection.createStatement();
201    String SQL1=
202      "SELECT DEPARTMENT_NAME AS depName\n" +
203      "FROM DEPARTMENTS\n" +
204      "WHERE DEPARTMENT_ID IN (SELECT DEPARTMENT_ID FROM EMPLOYEES)\n" +
205      "ORDER BY depName";
206    ResultSet resultSet1=statement1.executeQuery(SQL1);
207    ArrayList<String> depNameList=new ArrayList<>();
208    while(resultSet1.next())
209      depNameList.add(resultSet1.getString("depName"));
210    //részlegek feldolgozása
211    for(String depName: depNameList) {
212      //részleg dolgozói
213      String SQL2=
214        "SELECT FIRST_NAME || ' ' || LAST_NAME AS empName\n" +
215        "FROM EMPLOYEES E, DEPARTMENTS D\n" +
216        "WHERE E.DEPARTMENT_ID=D.DEPARTMENT_ID AND D.DEPARTMENT_NAME=?\n" +
217        "ORDER BY empName";
218      PreparedStatement preparedStatement2=connection.prepareStatement(SQL2);
219      preparedStatement2.setString(1, depName);
220      ResultSet resultSet2=preparedStatement2.executeQuery();
221      ArrayList<String> empNameList=new ArrayList<>();
222      while(resultSet2.next())
223        empNameList.add(resultSet2.getString("empName"));
224      //részleg létszáma
225      String SQL3=
226        "SELECT Count(EMPLOYEE_ID) AS empNo\n" +
227        "FROM EMPLOYEES E, DEPARTMENTS D\n" +
228        "WHERE E.DEPARTMENT_ID=D.DEPARTMENT_ID AND D.DEPARTMENT_NAME=?";
229      PreparedStatement preparedStatement3=connection.prepareStatement(SQL3);
230      preparedStatement3.setString(1, depName);
231      ResultSet resultSet3=preparedStatement3.executeQuery();
232      resultSet3.next();
233      int empNo=resultSet3.getInt("empNo");
234      //fa építése
235      String treeNodeText=depName+" ("+empNo+")";
236      DefaultMutableTreeNode treeNodeDepName=
237        new DefaultMutableTreeNode(treeNodeText);
238      treeRoot.add(treeNodeDepName);
239      for(String empName: empNameList) {
240        DefaultMutableTreeNode treeNodeEmpName=
241          new DefaultMutableTreeNode(empName);
242        treeNodeDepName.add(treeNodeEmpName);
243      }
244    }
245    connection.close();
246    tTree.setModel(treeModel);
247  }
```

4. ábra: Az 1. módszer megvalósítása

Mivel ez egy minta-adatbázis és egy hosszú Oracle tutorial start állapota, így hiányos – vagyis tárol olyan részlegeket, amelyekhez még nincsenek alkalmazottak hozzárendelve, mert ez egy későbbi import során történik meg az oktatóanyagban – így az üres részlegeket ki kell hagyni az eredménytáblából; erről a WHERE alparancs gondoskodik. A részlegek neveinek megfelelő sorrendje a SELECT parancs felelőssége. Minden részleg neve különböző. Minden alkalmazott csupán egyetlen részlegben dolgozik. A megvalósítás szempontjából nem lényeges, de minden alkalmazott neve is különböző.

A SELECT parancs futtatását követően kapott eredménytábla (resultSet1) egy szöveges oszlopból áll (alias depName), amely átmenetileg a depNameList generikus adatszerkezetbe kerül (4. ábra: 206-209. sor).

Ezt követően kezdődik a részlegek nevét tartalmazó generikus lista feldolgozása, (nem felelős a sorrendért). Ez koncepcionálisan három műveletből áll. Két lekérdező lépés: szükséges a részleg alkalmazottainak neve, valamint a részleg létszáma. Ezt követi a rendelkezésre álló adatokból a fa dinamikus felépítése. Ahol csak lehet, érdemes paraméteres lekérdező (PreparedStatement) utasítást alkalmazni, mert a típusellenőzés fordítási idejű, így kevesebb a hibalehetőség.
A lista feldolgozásának egy adott lépése (4. ábra: 211-244. sor) a fának egy részlegnév+ " "+(létszám) szöveget tároló csomópontját és az alatta elhelyezkedő alkalmazottak nevét sorrendben tartalmazó levélelemeket állítja elő:

- A lista feldolgozásának 1. lépéseként az aktuális részleg alkalmazottainak nevét kérdezi le sorrendben egy paraméteres SELECT parancs (SQL2). A paramétert a generikus listát (depNameList) bejáró for ciklus aktuális depName változója adja. A resultSet2 eredménytábla egy szöveges oszlopból áll, amely egy újabb generikus listába (empNameList) kerül (4. ábra: 213-223. sor). A részleg alkalmazottai nevének megfelelő sorrendje a SELECT parancs felelőssége.
- A lista feldolgozásának 2. lépéseként az aktuális részleg létszámát kérdezi le egy paraméteres SELECT parancs (SQL3), azonos paraméterrel. A resultSet3 eredeménytábla a Count aggregáló függvény miatt most egyetlen egész szám, amely az empNo változóba kerül (4. ábra: 225-233. sor).
- Végül a lista feldolgozásának 3. lépéseként először létrejön az aktuális részleghez tartozó fa csomópont (treeNodeDepName) a megfelelő szöveggel (treeNodeText). Ezután az aktuális csomópont a vizuális komponens mögötti treeRoot adatmodellhez hozzáadásra kerül. Ezután a generikus listát bejárva, az aktuális csomóponthoz megfelelő sorrendben hozzáadásra kerülnek az alkalmazottak nevei. A generikus lista nem felel a megfelelő sorrendért (4. ábra: 235-242. sor).

Két befejező művelet maradt hátra. Le kell zárni a megnyitott adatbáziskapcsolatot és gondolkodni kell arról, hogy az adatfeldolgozás során előállított fa adatszerkezet belekerüljön a vizuális komponens mögötti adatmodellbe, ezért fog megjelenni a GUI-n (4. ábra: 245-246. sor).

A 2. módszer

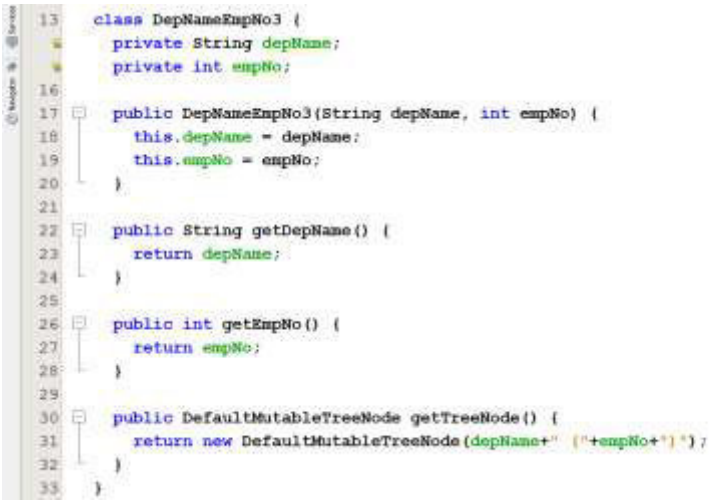
Az 1. módszert átgondolva kézenfekvő az ötlet a továbbfejlesztésre. Építve a már létező SQL1 és SQL2 lekérdező parancsokra, hagyjuk el az SQL3 parancsot és helyettesítsük mással. Az adott részleg létszámát az empNameList generikus listától is megkérdezhetjük (5. ábra: 279. sor).

```
249  private void method2() throws SQLException, ClassNotFoundException {
250    String url="jdbc:oracle:thin:@localhost:1521:xe";
251    String user="HR";
252    String password="hr";
253    Class.forName("oracle.jdbc.driver.OracleDriver");
254    DefaultMutableTreeNode treeRoot=new DefaultMutableTreeNode("Shipping firm");
255    DefaultTreeModel treeModel=new DefaultTreeModel(treeRoot);
       Connection connection=DriverManager.getConnection(url, user, password);
257    Statement statement1=connection.createStatement();
258    String SQL1=
259      "SELECT DEPARTMENT_NAME AS depName\n" +
260      "FROM DEPARTMENTS\n" +
261      "WHERE DEPARTMENT_ID IN (SELECT DEPARTMENT_ID FROM EMPLOYEES)\n" +
262      "ORDER BY depName";
263    ResultSet resultSet1=statement1.executeQuery(SQL1);
264    ArrayList<String> depNameList=new ArrayList<>();
265    while(resultSet1.next())
266      depNameList.add(resultSet1.getString("depName"));
267    for(String depName: depNameList) {
268      String SQL2=
269        "SELECT FIRST_NAME || ' ' || LAST_NAME AS empName\n" +
270        "FROM EMPLOYEES E, DEPARTMENTS D\n" +
271        "WHERE E.DEPARTMENT_ID=D.DEPARTMENT_ID AND D.DEPARTMENT_NAME=?\n" +
272        "ORDER BY empName";
273      PreparedStatement preparedStatement=connection.prepareStatement(SQL2);
274      preparedStatement.setString(1, depName);
275      ResultSet resultSet2=preparedStatement.executeQuery();
276      ArrayList<String> empNameList=new ArrayList<>();
277      while(resultSet2.next())
278        empNameList.add(resultSet2.getString("empName"));
279      String treeNodeText=depName+" ("+empNameList.size()+")";
280      DefaultMutableTreeNode treeNodeDepName=
281        new DefaultMutableTreeNode(treeNodeText);
282      treeRoot.add(treeNodeDepName);
283      for(String empName: empNameList) {
284        DefaultMutableTreeNode treeNodeEmpName=
285          new DefaultMutableTreeNode(empName);
286        treeNodeDepName.add(treeNodeEmpName);
287      }
288    }
289    connection.close();
290    tTree.setModel(treeModel);
291  }
```

5. ábra: A 2. módszer megvalósítása

A 3. módszer

Az 1. és 2. módszerekből közösen kiindulva készítsünk DepNameEmpNo3 nevű egyszerű POJO-t, amely az adott részleg nevét és létszámát képes tárolni. A szokásos gettereken kívül bízzuk rá annak felelősségét is, hogy a későbbi adatfeldolgozás során szükség esetén képes legyen visszaadni azt a szöveget, amit a részleghez kötődő fa csomópont megjelenít (6. ábra: 30-32. sor). Ilyen POJO-hoz alakítsuk át szükség esetén a lekérdező parancsokat, az átmeneti generikus listákat, módosítsuk az adatfeldolgozás lépéseit és a fa előállítását.



```
13  class DepNameEmpNo3 {
14    private String depName;
15    private int empNo;
16
17    public DepNameEmpNo3(String depName, int empNo) {
18      this.depName = depName;
19      this.empNo = empNo;
20    }
21
22    public String getDepName() {
23      return depName;
24    }
25
26    public int getEmpNo() {
27      return empNo;
28    }
29
30    public DefaultMutableTreeNode getTreeNode() {
31      return new DefaultMutableTreeNode(depName+" ("+empNo+")");
32    }
33  }
```

6. ábra: A 3. módszer POJO osztálya

Módosítani kell az SQL1 parancsot. Az lesz a feladata, hogy a részlegek nevét és létszámát kétoszlopos eredménytáblában (resultSet1) adja vissza. A Count aggregáló függvényhez kapcsolódik a GROUP BY alparancs. A megfelelő sorrend is a lekérdező parancs felelőssége (7. ábra: 302-308. sor). Az eredménytáblából építsünk depNameEmpNoList nevű DepNameEmpNo3 típusú generikus listát (7. ábra: 309-316. sor).

Ezt követően kezdődik a részlegek nevét tartalmazó generikus lista feldolgozása, (nem felelős a sorrendért). Ez koncepcionálisan három műveletből áll. Első a fa csomópont elkészítése. Második – lekérdező – lépés: szükséges a részleg alkalmazottainak neve. Ezt követi a rendelkezésre álló adatokból a fa csomópont leveleinek dinamikus felépítése (7. ábra: 317-334. sor). A lista feldolgozásának egy adott lépése a fának egy részlegnév+ " "+(létszám) szöveget tároló csomópontját és az alatta elhelyezkedő alkalmazottak nevét sorrendben tartalmazó levélelemeket állítja elő. A feldolgozandó generikus lista POJO, azaz DepNameEmpNo3 típusú. Ennek okos gettere (getTreeNode() függvény) megadja a fa csomópont szövegét (7. ábra: 318-319. sor).

```
293  private void method3() throws SQLException, ClassNotFoundException {
294    String url="jdbc:oracle:thin:@localhost:1521:xe";
295    String user="HR";
296    String password="hr";
297    Class.forName("oracle.jdbc.driver.OracleDriver");
298    DefaultMutableTreeNode treeRoot=new DefaultMutableTreeNode("Shipping firm");
299    DefaultTreeModel treeModel=new DefaultTreeModel(treeRoot);
       Connection connection=DriverManager.getConnection(url, user, password);
301    Statement statement1=connection.createStatement();
302    String SQL1=
303      "SELECT DEPARTMENT_NAME AS depName, Count(EMPLOYEE_ID) empNo\n" +
304      "FROM DEPARTMENTS D, EMPLOYEES E\n" +
305      "WHERE D.DEPARTMENT_ID=E.DEPARTMENT_ID AND "+
306        "D.DEPARTMENT_ID IN (SELECT DEPARTMENT_ID FROM EMPLOYEES)\n" +
307      "GROUP BY DEPARTMENT_NAME\n" +
308      "ORDER BY depName";
309    ResultSet resultSet1=statement1.executeQuery(SQL1);
310    ArrayList<DepNameEmpNo3> depNameEmpNoList=new ArrayList<>();
311    while(resultSet1.next()) {
312      String depName=resultSet1.getString("depName");
313      int empNo=resultSet1.getInt("empNo");
314      DepNameEmpNo3 depNameEmpNo=new DepNameEmpNo3(depName, empNo);
315      depNameEmpNoList.add(depNameEmpNo);
316    }
317    for(DepNameEmpNo3 depNameEmpNo: depNameEmpNoList) {
318      DefaultMutableTreeNode treeNodeDepName=depNameEmpNo.getTreeNode();
319      treeRoot.add(treeNodeDepName);
320      String SQL2=
321        "SELECT FIRST_NAME || ' ' || LAST_NAME AS empName\n" +
322        "FROM EMPLOYEES E, DEPARTMENTS D\n" +
323        "WHERE E.DEPARTMENT_ID=D.DEPARTMENT_ID AND D.DEPARTMENT_NAME=?\n" +
324        "ORDER BY empName";
325      PreparedStatement preparedStatement=connection.prepareStatement(SQL2);
326      preparedStatement.setString(1, depNameEmpNo.getDepName());
327      ResultSet resultSet2=preparedStatement.executeQuery();
328      while(resultSet2.next()) {
329        String treeNodeText=resultSet2.getString("empName");
330        DefaultMutableTreeNode treeNodeEmpName=
331          new DefaultMutableTreeNode(treeNodeText);
332        treeNodeDepName.add(treeNodeEmpName);
333      }
334    }
335    connection.close();
336    tTree.setModel(treeModel);
337  }
```

7. ábra: A 3. módszer megvalósítása

A lista feldolgozása során végrehajtódik a korábbi SQL2 parancs és előállítja az aktuális részleg alkalmazottainak nevét (a sorrendért felelős). A resultSet2 eredménytábla egy szöveges oszlopból áll, amelyet most nem tárolunk el újabb generikus listába, ehelyett röptében feldolgozzuk (7. ábra: 328-333. sor). Az eredménytábla feldolgozása során bővül az adott treeNodeDepName fa csomópont az adott alkalmazott nevét (treeNodeText) tároló levélelemmel (treeNodeEmpName).

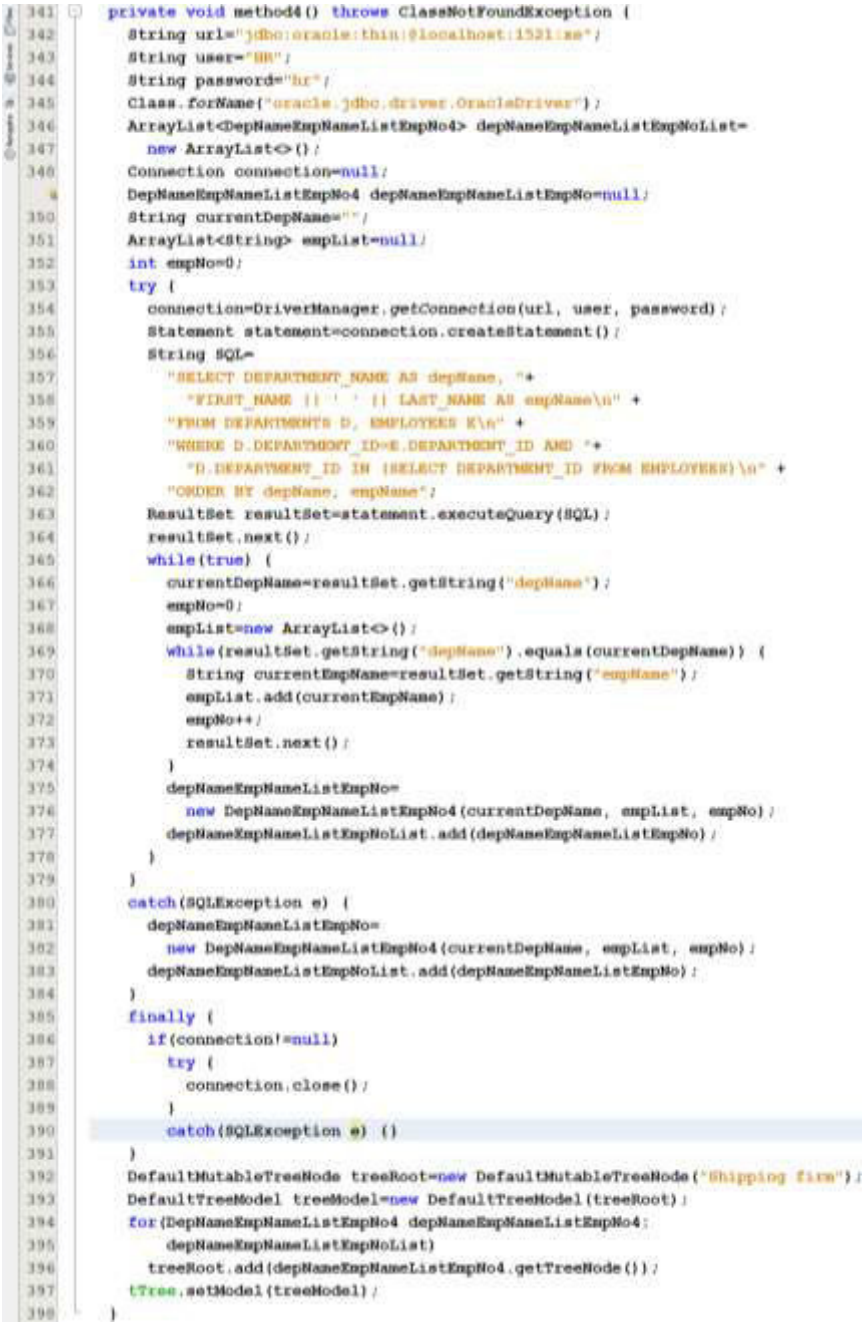
A 4. módszer

A 3. módszerből kiindulva készítsünk DepNameEmpNameListEmpNo4 nevű még okosabb POJO-t, amely az adott részleg nevét, a részleg összes alkalmazottjának nevét sorrendben és a részleg létszámát képes tárolni. Az alkalmazottnevek sorrendjéért ne tartozzon felelősséggel. Legyen képes visszaadni azt a DefaultMutableTreeNode osztályú objektumot, amely közvetlenül hozzáadható a vizuális komponens mögötti adatmodell fa adatszerkezetéhez (8. ábra: 47-52. sor) és tartalmaz egy fa csomópontot (treeDepName) és levélelemeket is (az empList-ből előállítva). Ilyen POJO-hoz alakítsuk át a már meglévő implementációt.

```
35  class DepNameEmpNameListEmpNo4 {
36    private String depName;
37    private ArrayList<String> empList=new ArrayList<>();
38    private int empNo;
39
40    public DepNameEmpNameListEmpNo4(String depName, ArrayList<String> empList,
41        int empNo) {
42      this.depName = depName;
43      this.empList = empList;
44      this.empNo = empNo;
45    }
46
47    public DefaultMutableTreeNode getTreeNode() {
48      DefaultMutableTreeNode treeDepName=
49        new DefaultMutableTreeNode(depName+" ("+empNo+")");
50      for (String empName: empList)
51        treeDepName.add(new DefaultMutableTreeNode(empName));
52      return treeDepName;
53    }
54  }
```

8. ábra: A 4. módszer POJO osztálya

A még okosabb POJO-nak nyilván további előnye is van, csökkenthető az SQL utasítások száma. Lehetnek hátrányok is, amelyek a programozási nyelvtől (akár veziószámtól) is függ(het)nek, például kivételkezelés. Az okos POJO-ból a generikus adatszerkezet felépítése kompromisszumokat követelhet, ráadásul több adatnak kell egyszerre rendelkezésre állnia, amikor meghívjuk a POJO konstruktorát (9. ábra: 375-376. sor).
Induljunk ki egyetlen SQL utasításból, amely előállít egy kétoszlopos resultSet eredménytáblát. Ez tartalmazza a részlegek nevét és az alkalmazottak nevét megfelelő sorrendben: részlegnevek ábécében, azon belül alkalmazottnevek ábécében (9. ábra: 354-363. sor). Így a lekérdező utasítások száma független az adatok számától. A megfelelő sorrend kiemelten fontos, hiszen erre épül a csoportváltás.



```
341  private void method4() throws ClassNotFoundException {
342    String url="jdbc:oracle:thin:@localhost:1521:xe";
343    String user="HR";
344    String password="hr";
345    Class.forName("oracle.jdbc.driver.OracleDriver");
346    ArrayList<DepNameEmpNameListEmpNo4> depNameEmpNameListEmpNoList=
347      new ArrayList<>();
348    Connection connection=null;
       DepNameEmpNameListEmpNo4 depNameEmpNameListEmpNo=null;
350    String currentDepName="";
351    ArrayList<String> empList=null;
352    int empNo=0;
353    try {
354      connection=DriverManager.getConnection(url, user, password);
355      Statement statement=connection.createStatement();
356      String SQL=
357        "SELECT DEPARTMENT_NAME AS depName, "+
358          "FIRST_NAME || ' ' || LAST_NAME AS empName\n" +
359        "FROM DEPARTMENTS D, EMPLOYEES E\n" +
360        "WHERE D.DEPARTMENT_ID=E.DEPARTMENT_ID AND "+
361          "D.DEPARTMENT_ID IN (SELECT DEPARTMENT_ID FROM EMPLOYEES)\n" +
362        "ORDER BY depName, empName";
363      ResultSet resultSet=statement.executeQuery(SQL);
364      resultSet.next();
365      while(true) {
366        currentDepName=resultSet.getString("depName");
367        empNo=0;
368        empList=new ArrayList<>();
369        while(resultSet.getString("depName").equals(currentDepName)) {
370          String currentEmpName=resultSet.getString("empName");
371          empList.add(currentEmpName);
372          empNo++;
373          resultSet.next();
374        }
375        depNameEmpNameListEmpNo=
376          new DepNameEmpNameListEmpNo4(currentDepName, empList, empNo);
377        depNameEmpNameListEmpNoList.add(depNameEmpNameListEmpNo);
378      }
379    }
380    catch(SQLException e) {
381      depNameEmpNameListEmpNo=
382        new DepNameEmpNameListEmpNo4(currentDepName, empList, empNo);
383      depNameEmpNameListEmpNoList.add(depNameEmpNameListEmpNo);
384    }
385    finally {
386      if(connection!=null)
387        try {
388          connection.close();
389        }
390        catch(SQLException e) {}
391    }
392    DefaultMutableTreeNode treeRoot=new DefaultMutableTreeNode("Shipping firm");
393    DefaultTreeModel treeModel=new DefaultTreeModel(treeRoot);
394    for(DepNameEmpNameListEmpNo4 depNameEmpNameListEmpNo4:
395        depNameEmpNameListEmpNoList)
396      treeRoot.add(depNameEmpNameListEmpNo4.getTreeNode());
397    tTree.setModel(treeModel);
398  }
```

9. ábra: A 4. módszer megvalósítása

Az előállított resultSet eredménytáblát egyszintű csoportváltás algoritmussal dolgozzuk fel (9. ábra: 364-378. sor). Az elv: amíg azonos nevű részleget találunk, addig adjuk hozzá a részleghez az alkalmazottakat. Mindezt addig, amíg el nem fogynak a részlegek (és persze az azon belüli alkalmazottak). Előkészítés: az adott részleg nevét tudni kell, az alkalmazottnevek listáját (empNameList) részlegenként üríteni kell, számolni kell/lehet az adott részleg alkalmazottait (empNo).

Az egyszintű csoportváltás standard algoritmusa indexelhető adatszerkezetre közismert, de itt – ebben a könyezetben és ebben az állapotban – egy resultSet eredménytábla feldolgozása közben nincs index és az eredménytábla sorainak számát sem adja vissza egyetlen metaadat sem. Az a koncepció, hogy az eredménytábla egyszer járható be és egyszer csak elfogy. Így jelentős kompromisszum, hogy a külső ciklus végtelen. A belső ciklus egy adott részleg feldolgozásáért felelős.

A belső ciklus vége után tudjuk bővíteni a depNameEmpNameListEmpNoList generikus, POJO típusú listát az aktuális depNameEmpNameListEmpNo objektummal. Az adatfeldogozás jellegéből adódóan kivétel keletkezik az eredménytábla utolsó rekordja utáni olvasás során (resultSet.next() hívásakor). Ezért – mintegy kvázi utófeldolgozásként – meg kell ismételni a külső ciklus végén lévő utasításokat (9. ábra: 375-377. sor) a keletkező SQLException osztályú kivételobjektumot elkapó catch ágban (9. ábra: 381-383. sor). Ha ezt nem tennénk meg – hanem például egyszerűen elnyelnénk a kivételt –, akkor a megjelenítendő fa struktúrából nem csupán az utolsó részleg utolsó alkalmazottja, hanem a teljes utolsó részleg az összes alkalmazottjával együtt kimaradna.
Érdemes áttekinteni a láthatóság, hozzáférhetőség felelősségét is. A POJO előállítása az egyszintű csoportváltás algoritmus felelőssége. Mivel a végtelen ciklus miatt ez nem oldható meg a try blokkban, így a depNameEmpNameListEmpNoList generikus listának „túl kell élnie" a try blokkot – hiszen még a catch-ben bővülni fog és csak az adatbáziskapcsolat lezárását követően épül a generikus lista POJO elemeiből a vizuális komponens mögötti fa adatszerkezet –, így a try blokk előtt kell deklarálni.
Az adatbáziskapcsolatot le kell zárni. Az adatfeldolgozás során biztosan keletkezik kivétel, így célszerű ezt a try-catch szerkezet finally blokkjában megtenni (9. ábra: 385-391. sor).
A megjelenítendő fa struktúra előállítása már nem kihívás, hiszen a generikus lista POJO objektuma mindent tud és direkt olyan objektumot képes visszadni, ami közvetlenül beszúrható a fába (9. ábra: 392-397. sor).
A megvalósítás adatközpontú, a feldolgozás elég összetett az eddigiekhez képest, valamint több kompromisszumot is megkívánt. Ez így – ebben a formában – nem ajánlott megoldási módszer.

Az 5. módszer

Az eddigi 4 módszerek koncepcióit, előnyeit, hátrányait átgondolva célszerűen adódnak ötletek. Például: tartsuk meg az egyszer lefutó SQL parancsot, használjunk POJO-t, de szabaduljunk meg a végtelen ciklustól.
A DepNameEmpName5 nevű POJO reprezentálja most a 4. módszer resultSet eredménytáblájának egy rekordját, azaz tárolja egy részleg nevét és egy dolgozó nevét (10. ábra).

```
56  class DepNameEmpName5 {
    private String depName, empName;
58
59      public DepNameEmpName5(String depName, String empName) {
60        this.depName = depName;
61        this.empName = empName;
62      }
63
64      public String getDepName() {
65        return depName;
66      }
67
68      public String getEmpName() {
69        return empName;
70      }
71  }
```

10. ábra: Az 5. módszer POJO osztálya

Ötlet: ha ebből a POJO-ból építünk generikus listát, akkor annak feldolgozása előtt lezárható az adatbáziskapcsolat és mivel a generikus lista elemszámát meg tudjuk kérdezni, így az egyszintű csoportváltás algoritmusa implementálható a klasszikus módon, két elöltesztelő ciklus alkalmazásával.
Tartsuk meg a 4. módszer SQL lekérdező parancsát (11. ábra: 409-415. sor). A resultSet feldolgozása során építsük fel a depNameEmpNameList generikus listát, amelybe DepNameEmpName5 osztályú objektumok kerülnek, azaz a fa felépítéséhez megfelelő sorrendben a részlegnév + alkalmazottnév adatok (11. ábra: 416-423. sor). A sorrendért a lekérdező utasítás felel.

Ezután lezárható az adatbáziskapcsolat (11. ábra: 424. sor). Ennek folyamatos nyitva tartása is lehet költség, így erre is lehet(ne) optimalizálni.
Most követhezhet a POJO objektumokat tartalmazó generikus lista feldolgozása egyszintű csoportváltás algoritmussal, amely során a megjelenítendő fa struktúra felépül (11. ábra: 425-440. sor). Az index 0-tól size()-1-ig mehet. A külső ciklus felel a részlegek léptetéséért, a belső ciklus pedig a részlegen belüli dolgozók léptetéséért. Mindez képletes és persze egymástól függő, hiszen az i++ postfix léptető utasítás a POJO attribútumai miatt mindkettőt lépteti, de a lekérdező parancs eredménytáblájának megfelelő sorrendje és a vezérlés együtt biztosítja a korrekt adatfeldolgozást. Kihasználjuk a fa csomópont Mutable tulajdonságát: a belső ciklus előtt üres megjelenítendő szöveggel jön létre treeNodeDepName nevű objektumként a fa csomópont, levelei röptében épülnek a belső ciklusban, de a szövege (userObject) csak a belső ciklus után kerül beállításra (11. ábra: 439. sor).

```
400   private void method5() throws SQLException, ClassNotFoundException {
401     String url="jdbc:oracle:thin:@localhost:1521:xe";
402     String user="HR";
403     String password="hr";
404     Class.forName("oracle.jdbc.driver.OracleDriver");
405     DefaultMutableTreeNode treeRoot=new DefaultMutableTreeNode("Shipping firm");
406     DefaultTreeModel treeModel=new DefaultTreeModel(treeRoot);
        Connection connection=DriverManager.getConnection(url, user, password);
408     Statement statement=connection.createStatement();
409     String SQL=
410       "SELECT DEPARTMENT_NAME AS depName, "+
411         "FIRST_NAME || ' ' || LAST_NAME AS empName\n" +
412       "FROM DEPARTMENTS D, EMPLOYEES E\n" +
413       "WHERE D.DEPARTMENT_ID=E.DEPARTMENT_ID AND "+
414         "D.DEPARTMENT_ID IN (SELECT DEPARTMENT_ID FROM EMPLOYEES)\n" +
415       "ORDER BY depName, empName";
416     ResultSet resultSet=statement.executeQuery(SQL);
417     ArrayList<DepNameEmpName5> depNameEmpNameList=new ArrayList<>();
418     while(resultSet.next()) {
419       String depName=resultSet.getString("depName");
420       String empName=resultSet.getString("empName");
421       DepNameEmpName5 depNameEmpName=new DepNameEmpName5(depName, empName);
422       depNameEmpNameList.add(depNameEmpName);
423     }
424     connection.close();
425     int i=0;
426     while(i<depNameEmpNameList.size()) {
427       String currentDepName=depNameEmpNameList.get(i).getDepName();
428       DefaultMutableTreeNode treeNodeDepName=new DefaultMutableTreeNode();
429       treeRoot.add(treeNodeDepName);
430       int empNo=0;
431       while(i<depNameEmpNameList.size() &&
432           depNameEmpNameList.get(i).getDepName().equals(currentDepName)) {
433         String currentEmpName=depNameEmpNameList.get(i).getEmpName();
434         treeNodeDepName.add(new DefaultMutableTreeNode(currentEmpName));
435         empNo++;
436         i++;
437       }
438       String treeNodeText=currentDepName+" ("+empNo+")";
439       treeNodeDepName.setUserObject(treeNodeText);
440     }
441     tTree.setModel(treeModel);
442   }
```

11. ábra: Az 5. módszer megvalósítása

A módszerek összehasonlítása, elemzése, tapasztalatok összefoglalása

Az elemzésnél számít a lekérdező parancsok száma és adott két jelölés: R a részlegek száma, k részleg létszáma. Az időigényt és a lépésszámot tekintsük azonosnak, vagy legalábbis egymással kölcsönösen megfeleltethetőnek.
Az 1. módszer esetén az időigény 1+2*R, a helyigény k*R, a bonyolultság triviális. Három különböző lekérdező parancs van (SQL1, SQL2 és SQL3). Az SQL2 és SQL3 lekérdező parancsok annyiszor hajtódnak végre, amennyi rekordot tartalmaz az SQL1 parancs eredménytáblája (R). Az SQL1 parancs egyszer fut le. Egy globális generikus lista (depNameList) és annyi lokális generikus lista (emp-

NameList) jön létre átmenetileg, amennyi elemet tartalmaz a depNameList (R). Ez jelentős terhet jelent az adatbázis-szerver és az automatikus szemétgyűjtő számára egyaránt. A koncepció ahhoz alkalmazkodik, hogy a megoldás közben milyen adatok keletkeznek.
A 2. módszer esetén az időigény 1+R, a helyigény k*R, a bonyolultság triviális. Két SQL parancs van (SQL1 és SQL2). Az SQL2 annyiszor fut le, amennyi rekordot tartalmaz az SQL1 parancs eredménytáblája (R). Sokat nyerünk azzal, hogy az SQL3 utasítás helyett az adatfeldolgozás során úgyis létrejövő átmeneti generikus listától – amely az adott részleg alkalmazottainak nevét tartalmazza – kérdezzük meg elemeinek számát. Ezzel R-rel redukálható az időigény. A koncepció még alkalmazkodik a megoldás közben keletkező adatokhoz, de már épít az „útközben" létrejövő adatszerkezet képességeire is.
A 3. módszer esetén az időigény 1+R, a helyigény R, a bonyolultság pedig az egyszerű POJO szintje. Továbbra is két SQL parancs van (SQL1 és SQL2). Az SQL2 annyiszor fut le, amennyi rekordot tartalmaz az SQL1 parancs eredménytáblája (R). Nyerünk a POJO okos getterével, mert ezzel alkalmazkodik a POJO-t tartalmazó generikus lista feldolgozása során előállítandó fa csomópont szövegigényéhez. Ezzel R-rel redukálható a helyigény. A koncepció a POJO-val új irányt vett: jobban épít az implementáció közben létrejövő adatszerkezetek képességeire.
A 4. módszer esetén az időigény 1, a helyigény k*R, a bonyolultságát pedig az összetett POJO szintje és összetett adatfeldolgozás együttesen határozza meg. Egyetlen SQL utasítás végrehajtását igényli a megoldás, így az ehhez kötődő időigény függetlenné vált az adatok mennyiségétől. A megoldás koncepcionálisan mégis rossz és nagyon adatmodell központú. A programozási nyelvből és a feldolgozás jellegéből együttesen adódóan szükséges a végtelen ciklus, amiből a feldolgozandó eredménytábla elfogyását követően keletkező kivételobjektum elkapásával tudunk kikerülni; azaz vezérlést végez a kivételkezelés, ami kifejezett lassú és nem ajánlott módszer. Fel kellett adnunk azt a kezdeti koncepciót is, hogy helyben nem kezelünk kivételt. Ráadásul az elvileg soha véget nem érő végtelen ciklus után építjük fel a fa adatszerkezetet – ezt még leírni is szörnyű. Viszont a fa felépítéséhez szükséges adatok mind rendelkezésre állnak a POJO típusú generikus listában, így az adatbáziskapcsolatot nem kell végig nyitva tartani – ez is lehet optimalizálási szempont. A helyigény is visszaesett az 1. és a 2. módszerek helyigényére. Ezt működő ipari megoldásnak tekinthetjük, de természetesen nem ajánlott módszer.
Az 5. módszert tekinthetjük az ajánlott megoldásnak. Időigénye 1, helyigénye R, bonyolultság szintjét az egyszerű POJO és az összetett adatfeldolgozás jelenti. Az összes eddigi módszer előnyét magába foglalja. Korrekt a láthatóság, a hozzáférhetőség, egyértelműek a felelősségi körök. Takarékos az erőforrásokkal, gyors, kevés memóriát igényel, hamar lezárja az adatbáziskapcsolatot, standard, elvárható adatfeldolgozási módszert alkalmaz.

Referenciák

[1] Oracle XE séma: http://web.tutscity.com/wp-content/uploads/2010/12/Tables-HR-Schema.png, 2015.05.10.
[2] Eckstein, R.: Java SE Application Design With MVC, http://www.oracle.com/technetwork/articles/javase/index-142890.html, 2015.05.10.
[3] Gábor Dénes Főiskola, ILIAS LCMS keretrendszer, Taneszköz tároló -> BSc/BA alapképzés -> Elosztott alkalmazások előadásvázlat, http://ilias.gdf.hu, 2015.05.10.
[4] Sommerville, I.: Szoftverrendszerek fejlesztése - Software Engineering, Panem Kft., 2007, ISBN 9789635454785
[5] Gamma, E. – Helm, R.: Programtervezési minták, Kiskapu Kiadó, 2004, ISBN 9789639301771